

 C# Corner
lar 8.0 - What's New And How To Upgrade

 vishnu

 C# Corner
PULSEBOOST HD

 FEEL THE BOOM!

[ASK A QUESTION](#) 

[CONTRIBUTE](#) 



ASP.NET MVC Life Cycle

In this article you will learn the ASP.NET MVC Life Cycle.



Rasmita Dash



Oct 16 2014



44

53

564.1k



[Download Free .NET & JAVA Files API](#)

In my initial days of learning MVC, I was curious about it's life cycle. But I found it a bit confusing. So I have documented my understandings. I hope this may help some one. This is a pure conceptual thing. To understand this, one needs to have a solid understanding of OOP concepts (especially about classes, interfaces, abstraction, inheritances and so on).

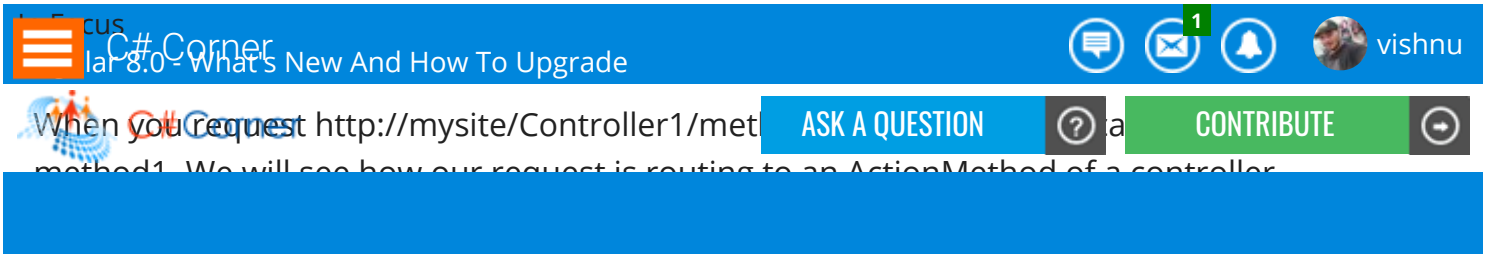
What happens in a normal ASP.NET application?

1. In an ASP.NET application each ASP.NET page inherits from System.Web.UI.Page that implements the IHttpHandler interface.
2. This interface has an abstract method ProcessRequest() and hence is implemented in the Page class. This method is called when you request a page.
3. The ProcessRequest() method takes an instance of HttpContext and is responsible for processing the request and generating the response.

So in an ASP.NET application it is so straight forward, you request a page with an URL like `http://mysite/default.aspx`. Then, ASP.NET searches for that page on the disk, executes the ProcessRequest() method, generates and renders the response. There is a one-to-one mapping between URL and physical page.

The ASP.NET MVC Process

In a MVC application, no physical page exists for a specific request. All the requests are routed to a special class called the Controller. The controller is responsible for generating the response and sending the content back to the browser. Also, there is a many-to-one mapping between URL and controller.



The procedure involved is,

1. An instance of the RouteTable class is created on application start. This happens only once when the application is requested for the first time.
2. The UrlRoutingModule intercepts each request, finds a matching RouteData from a RouteTable and instantiates a MVCHandler (an HttpHandler).
3. The MVCHandler creates a DefaultControllerFactory (you can create your own controller factory also). It processes the RequestContext and gets a specific controller (from the controllers you have written). Creates a ControllerContext. Passes the controller a ControllerContext and executes the controller.
4. Gets the ActionMethod from the RouteData based on the URL. The Controller Class then builds a list of parameters (to pass to the ActionMethod) from the request.
5. The ActionMethod returns an instance of a class inherited from the ActionResult class and the View Engine renders a view as a web page.


Now, let's understand in detail


1. Every ASP.NET MVC application has a RouteTable class. This RouteTable is responsible for mapping the MVC requests to a specific controller's ActionMethod.



Whenever the application starts, the Application_Start event will be fired and it will call the RegisterRoutes() with the collection of all the available routes of the MVC application as a parameter that will add the routes to the Routes property of the System.Web.Routing.RouteTable class. The Routes property is of type RouteCollection.



Global.asax

```
01. public class MvcApplication : System.Web.HttpApplication
02. {
03.     protected void Application_Start()
04.     {
05.         WebSecurity.InitializeDatabaseConnection("DefaultConnecti
06.         AreaRegistration.RegisterAllAreas();
07.         WebApiConfig.Register(GlobalConfiguration.Configuration);
08.         FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
09.         RouteConfig.RegisterRoutes(RouteTable.Routes);
10.         BundleConfig.RegisterBundles(BundleTable.Bundles);
```


C# Corner
 lar 8.0 - What's New And How To Upgrade





 vishnu

ASK A QUESTION
 
 CONTRIBUTE
 

```

02. {
03.     public static void RegisterRoutes(RouteCollection routes)
04.     {
05.         routes.IgnoreRoute("
06.         {resource}.axd/{*pathInfo}"); // Ignore route ending with axd
07.         routes.MapRoute(
08.             name: "Default", // route name
09.             url: "{controller}/{action}/{id}", // Url pattern
10.             defaults: new { controller = "Home", action = "Index"
11.         });
12.     }
  
```

The MapRoute() actually creates a default route. Adds all routes to the RouteTable and associates the RouteHandlers with the routes.

Now, all the mapped routes are stored as a RouteCollection in the Routes property of the RouteTable class.

NOTE

The RouteTable class has a Routes property that holds a collection of objects that derive from the [RouteBase](#) class. The RouteCollection class is derived from `Collection<RouteBase>`. Hence RegisterRoutes() is taking an object of RouteCollection.

When an ASP.NET MVC application handles a request, the application iterates through the collection of routes in the [Routes](#) property to find the route that matches the format of the URL requested. The application uses the first route that it finds in the collection that matches the URL. So the most specific route should be added first then the general ones.

2. Whenever you request an ASP.NET MVC application, the request is intercepted by the `UrlRoutingModule` (an HTTP Module) and:

- a. The `UrlRoutingModule` wraps up the current `HttpContext` (including the URL, form parameters, query string parameters and cookies associated with the current request) in an `HttpContextWrapper` object as below.

NOTE

The `HttpContext` class has no base class and isn't virtual and hence is unusable for testing (it cannot be mocked). The `HttpContextBase` class is a (from C# 3.5) replacement to `HttpContext`. Since it is abstract, it is mockable. It is concretely


C# Corner
 lar 8.0 - What's New And How To Upgrade





 vishnu

ASK A QUESTION

CONTRIBUTE

New HttpContextWrapper(HttpContext.Current).

```


02.     {
03.         HttpApplication application = (HttpApplication) sender;
04.         HttpContextBase context = new HttpContextWrapper(application)
05.         this.PostResolveRequestCache(context);
06.     }
  
```

Now, the module sends this `HttpContextBase` object to the `PostResolveRequestCache()` method.

- b. Based on the `HttpContextBase` object, the `postResolveRequestCache()` will return the correct `RouteData` from the `RouteTable` (that was created in the previous Step #1).
- c. If the `UrlRoutingModule` successfully retrieves a `RouteData` object then it creates a `RequestContext` object that represents the current `HttpContext` and `RouteData`.
- d. The `UrlRoutingModule` then instantiates a new `HttpHandler` based on the `RouteTable` and passes the `RequestContext` (created in Step c) to the new handler's constructor.
- e. For an ASP.NET MVC application, the handler returned from the `RouteTable` will always be an `MvcHandler`. This `MVCHandler` implements an `IHTTPHandler` interface and hence the `ProcessRequest()` method.
- f. Finally, it will call the `RemapHandler()` method that will set the `MVCHandler` just obtained to be the Current HTTP Handler.





```

01. public virtual void PostResolveRequestCache(HttpContextBase context)
02. {
03.
04.     RouteData routeData = this.RouteCollection.GetRouteData(context);
05.     if (routeData != null)
06.     {
07.         IRouteHandler routeHandler = routeData.RouteHandler;
08.         if (routeHandler == null)
09.         {
10.             throw new InvalidOperationException(string.Format(CultureInfo.InvariantCulture,
11.                 "SR.GetString(\"UrlRoutingModule_NoRouteHandler\"), new object[0]));
12.         }
13.         if (!(routeHandler == StopRoutingHandler))
  
```



C# Corner

lar 8.0 - What's New And How To Upgrade

vishnu

ASK A QUESTION

?

CONTRIBUTE

→

context.Request.RequestContext = requestContext;

```

17.
18.         if (httpHandler == null)
19.         {
20.             throw new InvalidOperationException(
21.
22.                 string.Format(CultureInfo.CurrentCulture,
23.
24.                     SR.GetString("UrlRoutingModule_NoHttpHandler"),
25.
26.                     new object[] { routeHandler.GetType() }));
27.         }
28.
29.         if (httpHandler == UrlAuthFailureHandler)
30.         {
31.             if (!FormsAuthenticationModule.FormsAuthRequired)
32.             {
33.                 throw new HttpException(0x191,
34.
35.                     SR.GetString("Assess_Denied_Description3"));
36.             }
37.             UrlAuthorizationModule.ReportUrlAuthorizationFailure(HttpContext
38.
39.                 }
40.             }
41.         }


```





3. MVCHandler is also inherited from the IHttpAsyncHandler hence implements the ProcessRequest() method. When MVCHandler executes, it calls the ProcessRequest() method that in turn calls the ProcessRequestInit() method.


The ProcessRequestInit() method creates a ControllerFactory and a Controller. The Controller is created from a ControllerFactory. There is a ControllerBuilder class that will set the ControllerFactory.

NOTE

By default it will be DefaultControllerFactory. But you can create your own ControllerFactory as well.


C# Corner
 lar 8.0 - What's New And How To Upgrade





 vishnu


 the Application_Start event in the global.as

ASK A QUESTION

?

CONTRIBUTE

→

Now, the NewFactory will be used instead of the DefaultControllerFactory .


The RequestContext and the name of the Controller (from the URL) will be ed to CreateController() method to get the specific Controller (that you have written).


Next, a ControllerContext object is constructed from the RequestContext and the controller using the method GetControllerInstance().

```

01. private void ProcessRequestInit(HttpContextBase httpContext, out ICon
02. {
03.     bool? isRequestValidationEnabled = ValidationUtility.IsValidati
04.     if (isRequestValidationEnabled == true)
05.     {
06.         ValidationUtility.EnableDynamicValidation(HttpContext.Cur
07.     }
08.     AddVersionHeader(httpContext);
09.     RemoveOptionalRoutingParameters();
10.     // Get the controller type
11.     string controllerName = RequestContext.RouteData.GetRequiredStr
12.     // Instantiate the controller and call Execute
13.     factory = ControllerBuilder.GetControllerFactory();
14.     controller = factory.CreateController(RequestContext, controlle
15.     if (controller == null)
16.     {
17.         throw new InvalidOperationException(
18.             String.Format(CultureInfo.CurrentCulture, MvcResources.Co
19.     }
20. }
21. public virtual IController CreateController(RequestContext requestCon
22. {
23.     if (requestContext == null)
24.     {
25.         throw new ArgumentNullException("requestContext");
26.     }
27.     if (string.IsNullOrEmpty(controllerName))
28.     {
29.         throw new ArgumentException(MvcResources.Common_NullOrEmp
30.     }
31.     Type controllerType = this.GetControllerType(requestContext, co
32.     return this.GetControllerInstance(requestContext, controllerTyp
33. }
  
```

- Our Controller inherits from the Controller class that inherits from ControllerBase that implements the Icontroller interface. The Icontroller interface has an Execute() abstract





C# Corner

lar 8.0 - What's New And How To Upgrade

ASK A QUESTION

CONTRIBUTE

1

vishnu

```

01. public abstract class Controller
02. {
03.     protected virtual void Execute(RequestContext requestContext)
04.     {
05.         // ...
06.         {
07.             throw new ArgumentNullException("requestContext");
08.         }
09.         if (requestContext.HttpContext == null)
10.         {
11.             throw new ArgumentException(MvcResources.Co
12.                 "requestContext");
13.         }
14.         VerifyExecuteCalledOnce();
15.         Initialize(requestContext);
16.         using (ScopeStorage.CreateTransientScope())
17.         {
18.             ExecuteCore();
19.         }
20.     }
21.     protected abstract void ExecuteCore();
22.     //Other stuffs here
23. }

```

- The ViewBag, ViewData, TempData and so on properties of the ControllerBase class is initialized. These properties are used for ing data from the View to the Controller or vice-versa or among action methods.
- The Execute() method of the ControllerBase class is executed that calls the ExecuteCore() abstract method. ExecuteCore() is implemented in the Controller class.

```

01. protected override void ExecuteCore()
02. {
03.     PossiblyLoadTempData();
04.     try
05.     {
06.         string actionName = RouteData.GetRequiredString("acti
07.         if (!ActionInvoker.InvokeAction(ControllerContext, ac
08.         {
09.             HandleUnknownAction(actionName);
10.         }
11.     }
12.     finally
13.     {
14.         PossiblySaveTempData();
15.     }
16. }

```



1



vishnu



C# Corner

ASK A QUESTION



CONTRIBUTE



d. The ExecuteCore() method then calls the InvokeAction() method of the ActionInvoker

as method parameters to the InvokeAction() method that is executed. Here the Controller objects viz. ControllerDescriptor and ActionDescriptor, that provide information on the controller (like name, type, actions) and Action (name, parameter and controller) respectively play a major role. Now you have your Controller name and Action name.

This controller class is something that you wrote. So one of the methods that you wrote for your controller class is invoked.

NOTE


controller methods that are decorated with the [NonAction] attribute will never be executed.





Finally It will call the InvokeAction method to execute the Action.


```

01. public virtual bool InvokeAction(ControllerContext controllerContext,
02. {
03.     if (controllerContext == null)
04.     {
05.         throw new ArgumentNullException("controllerContext");
06.     }
07.     if (string.IsNullOrEmpty(actionName))
08.     {
09.         throw new ArgumentException(MvcResources.Common_NullOrEmp
10.     }
11.     ControllerDescriptor controllerDescriptor = this.GetController
12.     ActionDescriptor actionDescriptor = this.FindAction(controllerC
13.     if (actionDescriptor == null)
14.     {
15.         return false;
16.     }
17.     FilterInfo filters = this.GetFilters(controllerContext, actionD
18.
19.     try
20.     {
21.         AuthorizationContext context = this.InvokeAuthorizationFi
22.         if (context.Result != null)
23.         {
24.             this.InvokeActionResult(controllerContext, context.
25.         }
26.         else
27.         {
28.             if (controllerContext.Controller.ValidateRequest)
29.             {

```



C# Corner
 lar 8.0 - What's New And How To Upgrade





 vishnu


C# Corner

IDictionary<string, object> parameterValues = this
 ActionExecut
 this.InvokeActionResultWithFilters(controllerContext

ASK A QUESTION

1

CONTRIBUTE

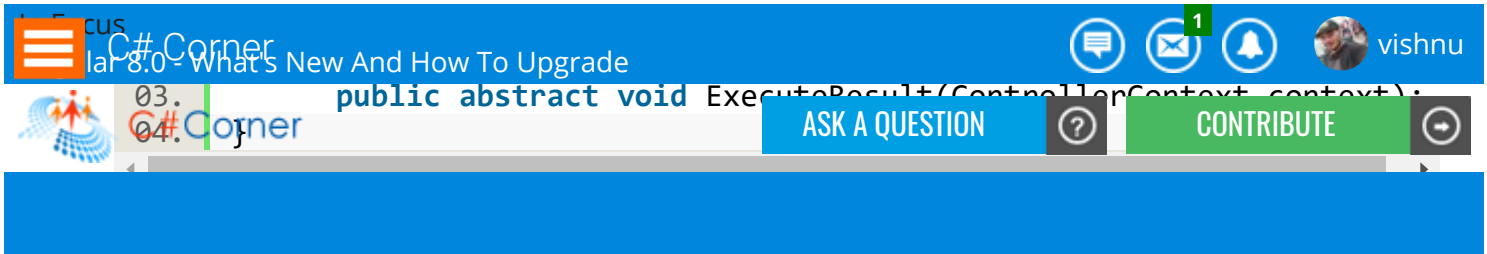
```

37.         catch (ThreadAbortException)
38.         {
39.             throw;
40.         }
41.         catch (Exception exception)
42.         {
43.             ExceptionContext context3 = this.InvokeExceptionFilters(c
44.             if (!context3.ExceptionHandled)
45.             {
46.                 throw;
47.             }
48.             this.InvokeActionResult(controllerContext, context3.Result
49.         }
50.         return true;
51.     }
  
```

5. The Controller returns an instance of ActionResult. The Controller typically executes one of the helper methods (mostly View()) that returns an instance of the ViewResult class, that is derived from the ActionResult class). Here's the list of classes that extend from the ActionResult class. You just need to call a specific Helper method to return the respective ActionResult.

Action Result	Helper Method	Description
ViewResult	View	Renders a view as a Web page.
PartialViewResult	PartialView	Renders a partial view, that defines a section of a view that can be rendered inside another view.
RedirectResult	Redirect	Redirects to another action method by using its URL.
RedirectToRouteResult	RedirectToAction RedirectToRoute	Redirects to another action method.
ContentResult	Content	Returns a user-defined content type.
JsonResult	Json	Returns a serialized JSON object.
JavaScriptResult	JavaScript	Returns a script that can be executed on the client.
FileResult	File	Returns binary output to write to the response.
EmptyResult	(None)	Represents a return value that is used if the action method must return a null result (void).

[Courtesy: [Controllers and Action Methods in ASP.NET MVC Applications](#)]



ViewResult is the most commonly used ActionResult. So let's discuss this.


The following happens after the ExecuteResult() method of ViewResult is called.





- ViewResultBase calls the FindView() method of the ViewResult class.
- The FindView() method of the ViewResult class returns an instance of the ViewEngineResult class.
- The Render() method of the ViewEngineResult class is called to Render the view using the ViewEngine.
- The Render() method internally calls the RenderViewPage() method that sets the master page location and ViewData.
- The response is rendered on client browser.


```


01. public virtual ViewEngineResult FindView( ControllerContext contr
02. {
03.     if (controllerContext == null)
04.     {
05.         throw new ArgumentNullException("ControllerContext");
06.     }
07.     if (string.IsNullOrEmpty(viewName))
08.     {
09.         throw new ArgumentException(MvcResources.Common_NullOrF
10.     }
11.     Func<IViewEngine, ViewEngineResult> cacheLocator = e => e.Fir
12.     Func<IViewEngine, ViewEngineResult> locator = e => e.FindView
13.     return Find(cacheLocator, locator);
14. }
15.
16. public virtual void Render(ViewContext viewContext, TextWriter wr
17. {
18.     if (viewContext == null)
19.     {
20.         throw new ArgumentNullException("viewContext");
21.     }
22.
23.     object obj2 = this.BuildManager.CreateInstanceFromVirtualPath(
24.     if (obj2 == null)
25.     {
26.         throw new InvalidOperationException(string.Format(Culture:
27.         new object[] { this.ViewPath }));
28.     }
29.     ViewPage page = (ViewPage) obj2;
30.     if (page != null)
31.     {


```

 C# Corner
lar 8.0 - What's New And How To Upgrade

 vishnu

 C# Corner

[ASK A QUESTION](#) 

[CONTRIBUTE](#) 

NOTE

The ViewPage class is derived from System.Web.UI.Page class. This is the same base class from which classic ASP.NET pages are derived.

The RenderView() method finally calls the ProcessRequest() method of the Page class that renders the view in the client browser.

References

- [ASP.NET MVC In-Depth: The Life of an ASP.NET MVC Request](#)
- [System.Web.Mvc](#)

[ASP.NET MVC](#)[ASP.NET MVC Life Cycle](#)[Learn MVC](#)

Rasmita Dash *TOP 500*

<https://www.c-sharpcorner.com/members/rasmita-dash>



480



3.1m



1

[View Previous Comments](#)



44



53